Interleaving in Systolic-Arrays: a Throughput Breakthrough

Interleaving in Systolic-Arrays: a Throughput Breakthrough

Giovanni Causapruno, Marco Vacca, Mariagrazia Graziano, *Member,* and Maurizio Zamboni

Abstract—In past years the most common way to improve computers performance was to increase the clock frequency. In recent years this approach suffered the limits of technology scaling, therefore computers architectures are shifting toward the direction of parallel computing to further improve circuits performance. Not only GPU based architectures are spreading in consideration, but also Systolic Arrays are particularly suited for certain classes of algorithms. An important point in favor of Systolic Arrays is that, due to the regularity of their circuit layout, they are appealing when applied to many emerging and very promising technologies, like Quantum-dot Cellular Automata and nanoarrays based on Silicon NanoWire or on Carbon nanotube Field Effect Transistors.
In this work we present a systematic method to improve Systolic Arrays performance exploiting Pipelining and Input Data Interleaving. We tackle the problem from a theoretical point of view first, and then we apply it to both CMOS technology and emerging technologies. On CMOS we demonstrate that it is possible to vastly improve the overall throughput of the circuit. By applying this technique to emerging technologies we show that it is possible to overcome some of their limitations greatly improving the throughput, making a considerable step forward toward the post-CMOS era.

Index Terms—Systolic arrays, CMOS, QCA, Molecular QCA, NanoMagnet Logic, NanoWire Field Effect Transistor, Interleaving

1 Introduction

SYSTOLIC Arrays (SAs) represented a solution frequently adopted to improve performance of computation-bound algorithms, heavily exploiting parallelism [1]. However, throughout the years, CMOS technological scaling has been sufficient to produce the required performance improvement, thus reducing the need of parallel architectures. In recent years SAs are back in the limelight for different reasons. First, there are applications, called "embarrassingly parallel applications" [2][3] in which parallelism is required to achieve the expected performance, and GPU-like processors are not always the best choice, compared to SAs [4][5]. Second, CMOS technological scaling is predicted to be near to its end [6][7] and parallelism can be exploited to improve throughput. Third, SAs are a great solution for new technologies (beyond-CMOS) [8][9][4][10][11]; in these technologies benefits in terms of speed and required area can be achieved only avoiding global interconnections [12][13], and SAs intrinsically fulfill this requirement.

It is then clear that SAs can be a great architectural solution, providing outstanding results, at least for certain kind of applications. A major limitation in the use of SAs is that the pipeline level, required to reduce the critical path and to increase frequency, could limit the throughput in presence of feedback loops. By increasing the level of pipelining higher clock frequencies are allowed [14], but problems arise due to signal synchronization, in particular in presence of feedback signals. As a consequence, to synchronize signals the circuits operations must be slowed down, leading to a throughput that might be lower than the throughput obtained with a lower pipeline level and a lower clock frequency. This is true both with CMOS [14] and with emerging technologies [4][9][11]. Furthermore, with the latter, the pipeline level is intrinsic in the technology and related to the circuit layout, so it can not be chosen by the designer, and it is usually extremely high [15][4][16].

One solution to this problem is to heavily exploit *interleaving* in order to increase throughput. With a principle similar to interleaving in processors out-of-order execution and, especially, to interleaved multithreading in modern CPUs, by accurately organizing input data delivery, SAs performance can be dramatically improved. Even though this approach has been adopted in some cases [17][18][19][10][9], a thorough and complete analysis based on a precise taxonomy of SAs has never been assessed in a rigorous way in previous works. In this paper we analyze this opportunity in details, showing to what extent interleaving can be applied, and how the possibility to use interleaving depends on the type of array and on the pipeline depth.

We introduce metrics and equations that can be used to understand benefits deriving when applying interleaving to a given circuit, and we provide rules to feed inputs in the right manner. The analysis is extremely general, and one could apply this method even without knowing the algorithm mapped in the array, but only the SA layout. The methodology then addresses the problem of long wire delays and analytically proposes interleaving, with a clear application-independent approach.

After an introduction on Systolic Arrays and Emerging Technologies (Section 2) and a detailed description of

Authors are with the Department of Electronics and Telecommunications, Politecnico di Torino, TO, I10129 Italy e-mail: mariagrazia.graziano@polito.it.

interleaving itself (Section 3), the taxonomy of SAs is introduced (Section 4). Then in Section 5, for each class of Systolic Arrays, a description of the optimization methodology is given: some parameters are explained and, depending on them, the number of data that can be interleaved and the delay between inputs is accurately defined. Finally in Sections 6 and 7 results on performance improvements are shown with some examples, using different levels of interleaving, both for CMOS arrays and for SAs designed in emerging technologies, such as QCA [13] and nanoarrays [20].

The solution here proposed represents a considerable step forward in the improvement of circuits performance, both for CMOS and for emerging technologies. This solution should always be adopted when possible, since it does not require extra hardware resources at the array level, but just a careful scheduling of inputs.

2 Background

In the following a brief summary on existing approaches to SAs is given both for CMOS and for emerging technologies. The aim is to let the reader better perceive the need for a systematic and ultimate definition of SAs and interleaving in SAs in the forthcoming scenario.

Systolic Arrays. To increase the performance of a computing system a common solution is to employ parallelism, using a large array of small processors. Systolic Arrays (SAs) were first introduced by Kung and Leiserson in 1978, who stated: "a systolic system is a network of processors which rhythmically compute and pass data through the system" [21]. Systolic Arrays are composed of Processing Elements (PEs) locally interconnected. Each PE receives data from neighbor cells or from outside and outputs result to the outside or to near PEs. Two are the main concepts at the basis of SAs: parallel computation (i.e. all PE work in the same way and in the same time on different data) and local transmission of data (i.e. there are not global signals). SAs are widely used in signal processing [22][23][24]; they are used also for algorithms in video processing (such as those for MPEG compression): for example, in [25] and [26] a SA for logarithmic search motion estimation is presented; by exploiting a bi-dimensional systolic architecture and *pipeline interleaving* the algorithm can be run 256 times faster than with a conventional linear array. SAs have been exploited also for image processing [27][28] and biological sequence comparison [29][5][30]: in [4] an overview of the different hardware solutions for biosequence analysis is carried out, showing that the best performance can be achieved adopting SAs, with a focus on nanotechnologies. Recently, automatic tools to translate algorithms to SAs for FPGAs have been explored [31]; reconfigurable arrays, that are not application-specific, have been introduced [32] as well.

SAs are not the only hardware method to improve performance of computation-bound algorithms. Throughout the years, technological scaling has allowed to increase operating frequencies and this has been enough to produce the required performance improvement; therefore, pure transliterations of the algorithm to hardware have been sufficient to achieve desired results thanks to the technology improvement. In recent years, with the development of $22\,nm$ technology for CMOS and the limited possibilities to further scale CMOS transistors [7][6] other ways to improve performance should be found. Highly parallel architectures such as Systolic Arrays are back in the limelight for this reason. Our study provides general techniques to improve throughput of SAs by using interleaving: in this way it is possible to increase the operating frequency and speed-up operations.

Emerging technologies. The approaching limits of CMOS technology, due to limitations such as leakage current and minimum fabrication sizes, paved the way for new technologies [33]. Quantum-dot Cellular Automata (QCA) is an emerging technology that is of particular interest for its low energy and small area requirements, as well as high frequencies achievable [34]. The basic component of a QCA design is a cell: it consists of six quantum dots in a square array coupled by tunnel barriers. Electrons are able to tunnel between the dots, but cannot leave the cell. If two excess electrons (or bunches of electrons) are placed in the cell, Coulomb repulsion will force the electrons to dots on opposite corners. There are thus two energetically equivalent ground state polarizations, as shown in Fig. 1.A, which can be labeled logic "0" and "1". An intermediate "NULL" state (Fig. 1.A) is necessary to switch dots from one state to the other. An external electric field is applied to the circuit to force the cell in the "NULL" state. When the field is removed cells reach a new stable state depending on neighbor elements [13]. There are two main implementation of the QCA principle: Molecular QCA, where molecules are used as base cell, that are studied for the high theoretical clock frequency that can reach (1 THz) [35][36][37] and magnetic QCA, also referred as Nano Magnet Logic (NML), where single domain nanomagnets are used as base cell [38][39][40], that are interesting for the low power consumption [41].

To propagate the information through the circuit a multiphase clock system is necessary [13]: many clock signals (for example 4 or 3) with different phases (Fig. 1.E) are applied to small areas of the circuit called clock zones (Fig. 1.D). The use of a multiphase clock system leads to an intrinsic pipelined behavior: Every group of 4 (or 3 [39]) consecutive clock zones has a delay of 1 clock cycle. The most important difference with CMOS case is that here the pipeline level is intrinsically related to the technology and it depends on the circuit layout [40]. The pipeline level is normally very high (hundreds of clock cycles) and it cannot be completely avoided but at least reduced [16]. While in pure combinational circuit this is not a problem because a pipelined structure allows to reach a very high throughput [42], in a sequential circuit this is not true. In these kind of circuits the results of a logic operation depend on the previous operations, but due to the intrinsic heavy

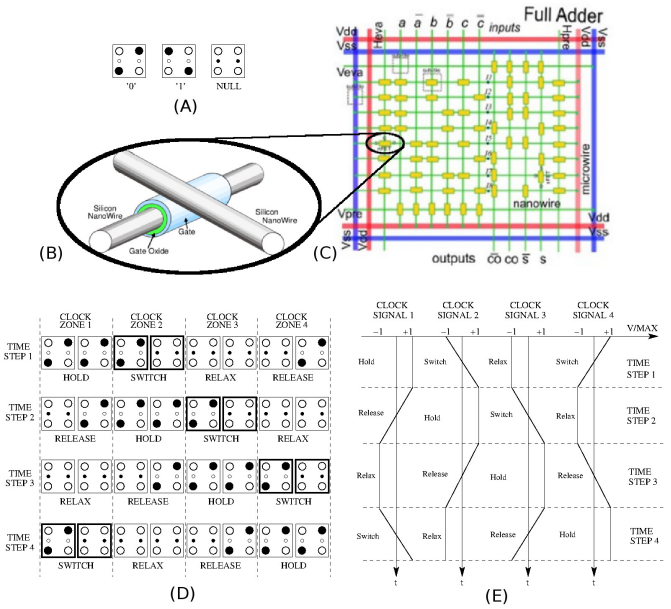


Fig. 1. **(A)** Elementary Quantum-dot Cellular Automata structures. **(B)** NanoWire Field Effect Transistor. **(C)** N-FET PLA-like structure. **(D)** QCA clock zones. **(E)** QCA clock phases.

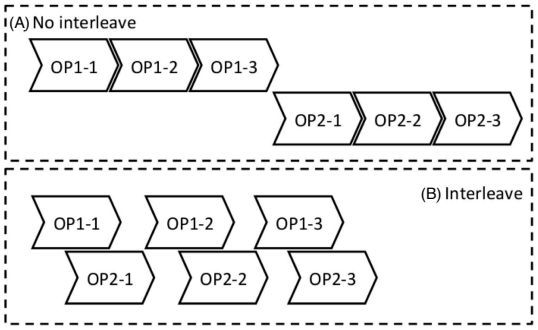


Fig. 2. Interleave example: two operations have to be executed, each requiring 3 steps; each step can be pipelined, and needs that the previous step is completed before executing. **(A)** If we don't interleave then steps cannot be pipelined. **(B)** With interleaving we can execute step 1 of operation 2 when the same step of operation 1 is still executing. This finally results in better performance.

pipelining, feedback signals need many clock signals to come back. As a consequence, in sequential circuits it is not possible to execute a logic operation at each clock cycle, because inputs must wait many clock cycles to synchronize with incoming feedback signals. Throughput is therefore reduced of N times, where N is the length in clock cycles of the longest loop in the circuit [43][16].

This is a well known problem in CMOS (i.e. the throughput reduction in RISC microprocessors due to conditional jumps), but it gains further importance with many emerging technologies that necessary have an high pipelining level. For example also Nanoarrays, based on Nanowires as interconnects and Nanowire Field Effect Transistors (N-FET) as active devices, potentially suffer from the same problem. N-FETs can be based on several type of materials. For example they can be based on two silicon nanowires separated by a small oxide layer (Fig. 1.B). Circuits are built in most of the cases in a PLA-like fashion (Fig. 1.C) organized in arrays of PLA-tiles and are driven by a dynamic 4-phase clock system [20][44]. No further details are given on this clock system because it is beyond the scope of this work and for space reasons as well, however its consequences are that every PLA-tile has a delay of one clock cycle. Since the size of the PLA-tile is limited, to build complex circuits many PLA-tiles must be connected together, leading to an intrinsic pipelining similar to the QCA case [4][12].

In recent years Systolic Arrays have been studied as a suitable target for nanotechnologies due to their regular layout and the lack of global interconnections. In QCA, for example, feasibility of a matrix multiplier Systolic Array has been demonstrated [8]; systolic and non-systolic design clock cycle count ratio for this architecture show the advantage of the adoption of SAs with QCA. In [11], an example of SA to execute Galois Field multiplication is reported. In [45] a reconfigurable PLA architecture for NML is presented and a reduction in area of about 10 times compared to a CMOS PLA is demonstrated. In [46] a NML SA for simplified convolution function is implemented.

In this paper the analysis and the suggested techniques are applied both to conventional and emerging technologies, and results are presented in section 6 and 7, respectively.

3 A SYSTEMATIC METHOD TO EXPLOIT SA: INTERLEAVING

To face the problem of deep pipelining and throughput reduction due to feedback signals, algorithmic solutions based on temporal correlation between data can be exploited. The solution here proposed is based on data interleaving, as a possible method to increase the circuit throughput. Usually, retiming technique is used to design Systolic Arrays and solve the constraints presented by feedback signals. In [47] authors present a systematic procedure to apply retiming to QCA SAs and thus optimize their throughput. This technique requires to take into account the topology of the circuit at an early stage of the design procedure. An important point of this approach is also that the optimization given by retiming could be bounded by clock layout constraints.

The interleaving approach is orthogonal to the retiming one. In the case of QCA, interleaving can be applied analyzing the circuit after the topology has been defined. In this way the designer does not need to take into account too carefully the circuit pipelining level during design phase, but he only has to understand how many operations can be interleaved. Moreover, the method we introduce can be applied also to CMOS SAs: each PE can be highly pipelined to reduce the critical path

and increase operating frequency; the high pipeline level does not affect the throughput if we adopt interleaving of data.

Interleaving is in general a way to arrange data in a non-contiguous way. Imagine that N operations must be executed; each of them requires a number of steps s. For example consider a matrix multiplication in which each element of the resulting matrix is evaluated through a number of multiplications and additions. One approach is to execute these operations one after the other, requiring sN steps, as shown in Fig. 2.A; in this case input data are passed in-order to the computational structure. However, a different approach is possible: instead of executing operations one after the other, they are interleaved so that steps are executed one after the other. This means that the order is: step 1 of op 1, step 1 of op 2, ..., step 1 of op N, step 2 of op 1 and so on. Hence, input data are passed to the structure in a interleaved manner (Fig. 2.B). The advantage of this approach is given by the fact that there is no data-dependency between successive interleaved steps. For example it is possible that for one operation step i requires step $i-1$ to be completed. In case steps are executed in order, they cannot be pipelined, because the execution must be stopped to wait for previous step to be completed before starting the new one. Using instead interleaving, while the circuit is waiting for the $i-1$ step, another step of a different operation can be executed because there is no data-dependency. In this way the pipe is always full and the throughput is maximized.

The benefits of interleaving (often referred as *pipeline interleaving*) have been analyzed in literature, especially in the case of digital filters [17][48]: internal feedbacks in digital filters negate the most obvious ways of improving performance, that is pipelining. In fact, recursive systems cannot be pipelined at an arbitrary level by simply inserting latches; the problem is solved by changing the internal structure of the algorithm to create additional logic delay operators inside the recursive loop, which can then be used for pipelining. The potential given by interleaving is exploited also in Digital Signal Processors (DSP); in [18] and [49] a system level solution starting from the algorithm and taking into account programmability of DSP is analyzed. Some works have been presented to transform algorithms to exploit interleaving: in [19] a method that selects and transforms a systolic algorithm into a parallel algorithm with high granularity is presented; this method returns the exact code to run on each processing element of the multicomputer system.

The approach used in this paper is different from those described above: we present here the method of interleaving for Systolic Arrays, exploiting it at two different levels:

1) PE level: the processing element must be adapted to exploit interleaving, changing its architecture;
2) SA level: input data must be transmitted in a precise order given by the interleaving level that can be achieved inside each PE and by the communication between PEs.

In the following we discuss the deployment of the two above mentioned approaches on general SAs, formalizing their possible properties and parameters, and explaining the possible solution space.

4 Taxonomy of Systolic Arrays

Herein we give a rigorous taxonomy of SAs in order to distinguish the possible topologies and characteristics. This taxonomy will be used throughout the whole paper.

Systolic Arrays can be divided into three main classes: those With cells that have an Internal Loop (herein **WIL**), those WithOut Internal Loop (herein **WOIL**) and those With External Loop (herein **WEL**). The former can be further split in systolic arrays that Store results in the cells (**WIL-S**) and systolic arrays where the partial result is Passed Through the cells to obtain the final value (**WIL-PT**). In the following four examples are reported to clarify the meaning of these definitions.

WOIL Example. Consider a problem of matrix multiplication: let $A = (a_{ik})$ and $B = (b_{kj})$ be two rectangular matrices of order $N_1 \times N_3$ and $N_3 \times N_2$ respectively. Their product, matrix $C = A \times B$, $C = (c_{ij})$ can be obtained according to the following formula:

$$c_{ij} = \sum_{k=1}^{N_3} a_{ik} \cdot b_{kj}, \quad i = 1, 2, \ldots, N_1 \quad j = 1, 2, \ldots, N_2 \quad (1)$$

This algorithm can be mapped to a SA with the structure shown in Fig. 3.A, which comprises a matrix of PEs. Each PE stores one element of matrix B; elements of the resulting matrix C are evaluated through the cells of one column, as shown in Fig. 3.A. Each cell will compute a multiplication between the incoming operand and the stored one, and will add the result to the partial c_{ij} coming from the upper cell; this is an example on WOIL Systolic Array, since no loop is required inside each PE.

WIL-S example Kung [1] has shown a systolic array for matrix multiplication that has cells with internal loop and each element of the resulting matrix is stored in a cell of the array, which is a WIL-S array, shown in Fig. 3.B. Each PE will receive a couple of input data, multiply them and add the result to the partial sum internally stored in the loop. The final value of each element c_{ij} of the resulting matrix will be generated from each PE.

WIL-PT example. One example of WIL-PT systolic array, made of cells with internal loop and results passed through the cells, can be derived for the evaluation of finite difference derivatives. Given one function $f(x)$ and the length of the step h, the derivative of the function can be approximated with the incremental ratio in equation (2):

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (2)$$

Signal Flow Graph (SFG) [1] for the problem is shown in Fig. 3.C. The resulting hardware implementation is

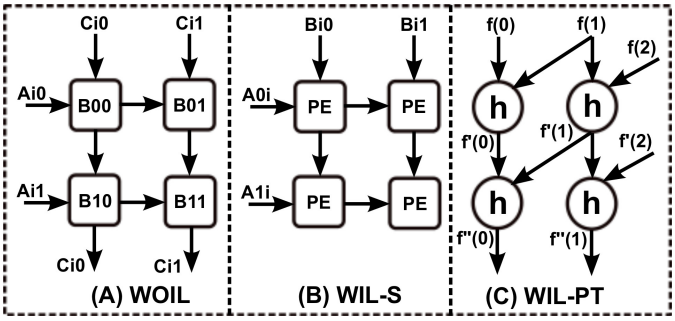


Fig. 3. **(A)** WOIL Systolic Array example: 2×2 matrix multiplier. Each PE executes a multiplication between inputs and adds the result to the partial result coming from previous PE. **(B)** WIL-S Systolic array example: 2×2 matrix multiplier with cells with internal loop and results stored in each cell. **(C)** Signal Flow Graph for finite difference derivative; each element must execute the difference between the two inputs and divide the result by h, to implement equation (2).

obtained shrinking the SFG along the vertical axis. In the derived SA, the local derivative evaluated by a cell is re-used by the same cell for the next-order derivative, and passed to the cell on its left.

WEL example. Lastly, we consider WEL systolic arrays: in this array the last cell of a line is connected backward to the first cell of the same line, producing an "external loop". An example of this class is the array that can be derived for Galois Field Multiplication, like the one presented in [11]. This external loop must be taken into account to synchronize incoming inputs from outside with values coming back from the loop.

In next sections each of these structures is analyzed with the aim of optimizing the usage of the cell, exploiting the pipelined structure of the elements of the cell to interleave different operations; the examples seen so far are used to clarify the explanation in the following sections.

5 Optimization of Systolic Arrays

5.1 WOIL Systolic Arrays

5.1.1 Definition

In this class of systolic arrays the results to be evaluated are passed through the cells of the array, which have no loop inside; the systolic array for matrix multiplication shown in Fig. 3.A, with each cell storing one element of matrix B, is part of this class.

In general the cell can be described, as shown in Fig. 4.A, as made of N different blocks, each requiring d_i clock cycles to be completed, $i = 1, 2, \ldots, N$. Some of them (for example from block $J+1$ to N) are along the path that connects the cells each other (called P), and their total delay is D in equation (3):

$$D = \sum_{i=J+1}^{N} d_i \tag{3}$$

while the others (from 1 to J) work on input data and on stored ones, and are not interested by partial results.

5.1.2 Optimization with interleaving

Let us consider the case in which the N blocks cannot be pipelined; each of them requires d_i cycles to complete the operation and during these cycles no new inputs can be given; hence, each of them can receive valid inputs every d_i cycles (notice that if a block can be pipelined in s stages, it can be thought as s different blocks, each with its own delay). The output rate of each block is the same of the input one, so, to match the delay condition for all the blocks of the processing element, input data should be given at least every K cycles: $K = \max\{d_i\}$ with $i = 0, 1, \ldots, N$. The throughput that can be achieved is $1/K$; however, without exploiting the intrinsic pipelined nature of the cell (where each block is a new stage of pipe), data would be given at the end of the whole computation, hence the throughput would be $1/\sum_i d_i$; it is then clear that a speed-up of $\sum_i d_i/K$ can be in principle achieved. As shown in Fig. 4.A, one cell receives data from previous cell at time T_0 and will produce the output at time $T_0 + D$; external inputs are given every K cycles; the delay between one cell and the successive is D, hence the external inputs to the following cell shall also have a delay of D with respect to the corresponding inputs in the previous cell. In other words, first processing element can be fed with input data at time 0, K, $2K$, ...; second processing element instead is fed with data at time D, $D+K$, $D+2K$, ...

5.1.3 Optimization with exact synchronization of inputs

Consider a two-dimension WOIL systolic array, for example for matrix multiplication, as shown in Fig. 3.A. Each cell will receive one input value a_{ik} from left (given to the boundary cell and locally transmitted through processing elements), will multiply it with the stored value in the cell b_{kj}, and will add the result with the partial result coming from the upper cell $c_{ij}(k-1)$ to obtain $c_{ij}(k)$; the structure of the PE is shown in Fig. 4.B. This process requires that all the cells are preloaded with the values of one of the two matrices to multiply, and this could require a large overhead, highly affecting performance.

What is presented hereinafter is a solution to this problem, that requires to give also the second input from outside during computation. This value must be valid every time a new computation is executed in the cell in order to be ready exactly as it would be if the data were internally stored as in the original solution. This means that the value must be transmitted from outside as many times as the cell requiring that input will be used. For example, in the case of 3×3 matrix multiplication, shown in Fig. 3.A, each value of b_{kj} must be transmitted 3 times.

Exact synchronization can be achieved inserting in every processing element a shift register to pass the input to the next cell (Fig. 5); call L the length of this shift register, M the number of cells in each line of the systolic

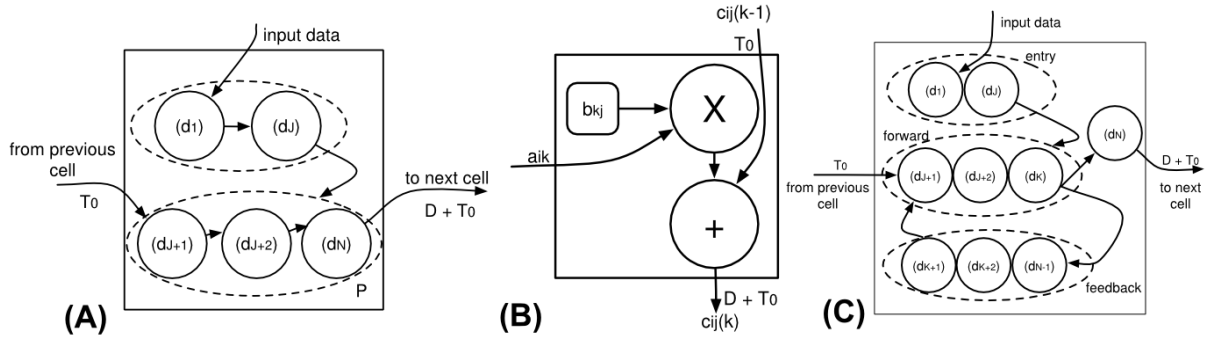


Fig. 4. **(A)** WOIL cell: d_i, $i = 0, 1, \ldots N$ is the delay of each block in clock cycles; blocks from 1 to J work on input data and on stored ones, while blocks from $J + 1$ to N are along the path P that connects the cells each other and their total delay is D. **(B)** WOIL cell for matrix multiplication: input value a_{ik} and stored one b_{kj} are multiplied and summed to the partial result, obtained after $k - 1$ steps, $c_{ij}(k - 1)$; the result is $c_{ij}(k)$. The delay of the path between cells is D. **(C)** WIL cell: this PE is made of 4 parts: an entry section, the forward and feedback parts of the loop, and the output section. Data coming from previous PE can enter at any stage of the cell.

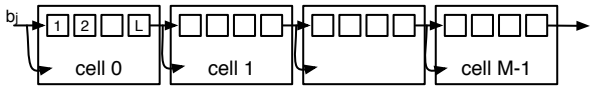


Fig. 5. SA row adapted for exact synchronization: input data are pumped in from left and passed through cells through shift registers, of length L.

array, N the number of successive inputs given to each cell. Values of b_j, $j = 0, \ldots M-1$ are fed through the input in the leftmost cell, and travel through the chain of shift registers. The condition to guarantee feasibility of exact synchronization is that there are not two different values that must be given at the input of the chain of shift registers at the same time (that would be impossible). Inputs to the first cell are given at time 0, K, $2K$, ...; inputs to the second cell must be given at time D, $D+K$, $D+2K$, ..., and, considering that they must pass through the shift register of the first processing element, that requires L cycles, they must be fed at the beginning of the chain at time $D - L$, $D + K - L$, $D + 2K - L$, ... The whole set of expressions extracted must be analyzed to ensure that there are not two expressions representing the same cycle; this can be expressed by formula (4):

$$mD + nK \neq mL \tag{4}$$

$$\forall\, m \in [0, M-1],\ n \in [-N+1, N-1]$$

where L and D are design parameters, and cannot be changed once the systolic array has been implemented. K instead, represents the frequency at which inputs are fed, and can be varied. In order to verify equation (4) it is hence possible to increase K; this would be often needed when $K < M$: in fact, K represents the distance between two occurrences of the same value (that goes in the same cell); in this period of time all values for the other cells must be fed in, and this is possible only if $K \geq M$. This condition gives also indication on how to design systolic arrays for exact synchronization: if the number of cells must be high, it is convenient to have big operations inside each processing element that require many cycles. There are other cases when formula (4) is not verified (meaning that there are two inputs that should be given at same time); in these cases increasing K of 1 could solve the problem. However, there are cases, such as $D = L$, in which the exact synchronization approach can never be adopted. We have analyzed the feasibility of this method for values of D from 1 to 20 and L from 1 to 10; it resulted that nearly 93% of cases are feasible, even though in $1/3$ of them it is required to increase the value of K of 1 (starting condition is $K = \max\{d_i, M\}$, $i = 1, 2, \ldots)$, leading to a decrease in performance of $K/(K+1)$, that is in general smaller than the overhead due to the preload of values in the cells. In fact, to preload values in the array it is necessary to input them from boundaries in reverse order (first the one that goes in last cell of the row), with a distance L each from the other. At the same time an "enable" signal for the register that stores the value is sent. This is repeated every time a new operation is started, and having N operations the total overhead due to preload is approximately $T_{ov}^{(stored)} = N(M-1)L$. On the other hand, if we use exact synchronization, K could be increased from the original value up to $M + 1$, and considering that this increase reflects on each input, the overhead would be $T_{ov}^{(exact)} = (M+1)N$. Comparing $T_{ov}^{(stored)}$ with $T_{ov}^{(exact)}$, it comes out that every time $M > (L+1)/(L-1)$ it is convenient to use the exact synchronization method. Having integer values for L it is clear that when $L = 1$ preload must be used; for $L = 2$ preload is convenient only if $M < 3$, and for higher values of L exact synchronization should be always used.

5.2 WIL-S Systolic Arrays

5.2.1 Definition

A cell with internal loop is shown in Fig. 4.C. It is made of 4 parts: an entry section, made of blocks numbered

from 1 to J; the forward part of the loop, made of blocks from $J+1$ to K, the feedback part of the loop, made of blocks from $K+1$ to $N-1$; the output block, called N. Each of these blocks is associated to a delay d_i, $i = 1, 2, \ldots, N$. Let us call T_e the total delay of the entry block, T_{ff} the delay of the forward side of the loop, T_{fb} the delay of the feedback in the loop and T_o the output delay as in the following relations:

$$T_e = \sum_{i=1}^{J} d_i \quad T_{ff} = \sum_{i=J+1}^{K} d_i \quad T_{fb} = \sum_{i=K+1}^{N-1} d_i \quad T_o = d_N \tag{5}$$

Input data coming from outside enter in the first block, while data coming from the neighbor processing element can enter at any stage of the cell. This cell can be part of a systolic array where each result is finally stored in the cells of the array (WIL-S), like in the example of matrix multiplication shown in Fig. 3.B; or it can be part of an array where local result are re-used in the cell and also passed to neighbor cells (WIL-PT), like in the example of finite difference derivative. These cells usually have also a shift register used to pass inputs to neighbor cells, whose delay is L, and that is not shown in Fig. 4.C for sake of simplicity.

The example of array for matrix multiplication reported in Fig. 3.B is part of WIL-S class; the cell for this operation is composed of a multiplier between the two incoming operands, a and b, passed to other cells through the internal shift registers; $a \times b$ is summed with the result stored inside the loop; a control signal *ctrl* is used to de-activate the input from the feedback at first iteration. Fig. 6 shows also the line to output the result once it has been evaluated; this line however is never used during computation of the resulting matrix. According to the nomenclature previously described, the delay of the multiplier will be T_e and the delay of the adder T_{ff}, while the feedback loop itself will require T_{fb} cycles.

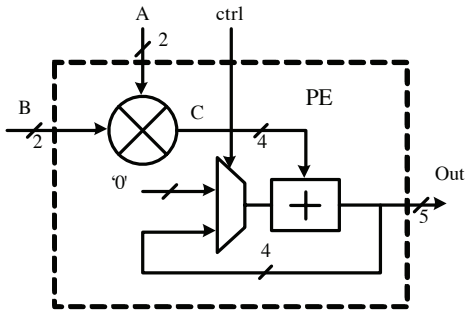


Fig. 6. Cell with internal loop used for matrix multiplication as shown in [11]. A and B are multiplied and added to the value stored in the loop. "ctrl" signal is used to select input '0' from the multiplexer for the first addition.

5.2.2 Optimization with interleaving

In order to match timing of inputs with delay of the feedback, inputs must be given every $T_{loop} = T_{ff} + T_{fb}$ cycles, that is the total time of the feedback loop. However, given the intrinsic pipelined nature of the structure, we can improve performance and usage of the cell giving inputs every $K = max\{d_i\}$, as done for the cells without loop. Every K cycles a new operation can be started, and in this way N different operations can be interleaved, being $N = T_{loop}/K$ (integer division). After T_{loop} cycles, the second set of inputs is fed; it will be multiplied and added to the value previously evaluated and stored in the feedback loop. When T_{loop} is not a perfect multiple of K, the remainder of the division, called R, must be taken into account: after N operations have been started, the following one must start with a delay of $K + R$ with respect to the previous, so to have synchronization with the result coming from the loop. R represents a number of "stalls" that must be inserted between one set of N inputs and the following set. Consider the following example: $T_e = 3$, $T_{ff} = 3$, $T_{fb} = 10$; it is possible to interleave $T_{loop}/K = 13/3 = 4$ operations, inserting a stall ($R = 1$) after each set of 4 inputs. This cell can be used for example to evaluate the elements of a 2×2 matrix, and the time of evaluation is reported in Table 1, showing also the stall cycle required to match data at input of the adder.

TABLE 1
Interleaved inputs to a WIL-S PE.

Time	Inputs	Result
0	a_{00}, b_{00}	$a_{00} \times b_{00}$
3	a_{00}, b_{01}	$a_{00} \times b_{01}$
6	a_{10}, b_{00}	$a_{10} \times b_{00}$
9	a_{10}, b_{01}	$a_{10} \times b_{01}$
12	stall	stall
13	a_{01}, b_{10}	$a_{00} \times b_{00} + a_{01} \times b_{10} = c_{00}$
16	a_{01}, b_{11}	$a_{00} \times b_{01} + a_{01} \times b_{11} = c_{01}$
19	a_{11}, b_{10}	$a_{10} \times b_{00} + a_{21} \times b_{10} = c_{10}$
21	a_{11}, b_{11}	$a_{10} \times b_{01} + a_{11} \times b_{11} = c_{11}$

With respect to the normal usage of this kind of cell, it is possible to evaluate in the same time N different operations, having an increase in performance of N. The number of stalls represents a factor of performance decrease, that must be carefully analyzed at design time; consider the following example: $T_{ff} = 20, T_{fb} = 6, K = 9$; it will result in $N = 2$ and $R = 8$. In this case it would be favorable to increase of 1 cycle the delay of the feedback, so to have 3 possible interleaved operations, and no stalls. As far as the global array is concerned, processing elements work independently each from the others, and the only connections are the shift registers to pass inputs from one cell to another. Being L the length of the shift register in each cell, inputs must be given with the same rule for all cells (number of interleaved operations and stall cycles), but starting at cycle $S_i = m_i \times L$; m_i is the Manhattan distance between cell i and the top-left one (if we consider data moving from top to bottom and left to right like in Fig. 3.B).

5.3 WIL-PT Systolic Arrays

5.3.1 Definition

Fig. 7 shows the systolic array that is directly obtained by shrinking the signal flow graph of Fig. 3.C along the vertical axis. The structure of the processing element is shown in Fig. 8.A. Each cell receives two values of the function, say $f(x_1)$ and $f(x_1+h)$, and evaluates the finite difference that approximates the derivative as expressed in formula (2):

$$f'(x_1) \approx \frac{f(x_1+h) - f(x_1)}{h}$$

Notice that the value of h is stored in the cell and it is the same for all cells. At next step, storing $f'(x_1)$ in the loop, and receiving $f'(x_1+h)$ from the neighbor cell, it will evaluate $f''(x_1)$, and so on. It is required that the rightmost cell receives at every step the derivative of any order of the boundary point (x_n+h).

5.3.2 Optimization with interleaving

Usage of this processing element can be optimized following the same rules given for the previous case: $N = T_{loop}/K$ (integer division) and $R = T_{loop}\%K$ (remainder) indicate how many operations can be interleaved and how many stall cycles must be given between sets of inputs, respectively. For the example of the cell for finite difference derivative, we have $T_e = 0$ and T_{ff} equal to the sum of the delay of the subtracter and the multiplier. At first cycle all cells can be fed in with values of the function; however, to respect exact timing of operations, cells have to be fed according to the unbalance that there is between the feedback loop (T_{fb}) and the path for the local result to be passed to neighbor cell (T_p). Consider two cells only: one will receive inputs from the external (independent cell), the other must receive the value calculated by the neighbor cell (dependent cell). If the delay of the feedback is longer than the one for forwarding data between cells, the dependent cell will start computation before, and the independent one will be fed with input values only when the delay mismatch has been covered, that is, $T_{fb} - T_p$. In the other case, if the forward path is longer, then the independent cell will begin computation earlier and the dependent cell will be fed with values only $T_p - T_{fb}$ cycles later. This analysis can be enlarged to a bigger array, where there is only one independent cell (the rightmost one in our example), and all the others are dependent cells (each cell is dependent from the one at its right); calculations must start from the rightmost or leftmost cell according to the unbalance between the two paths.

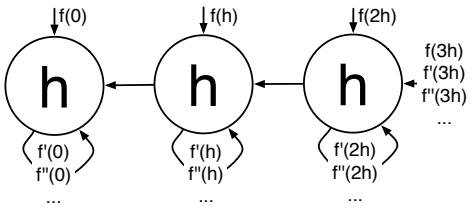
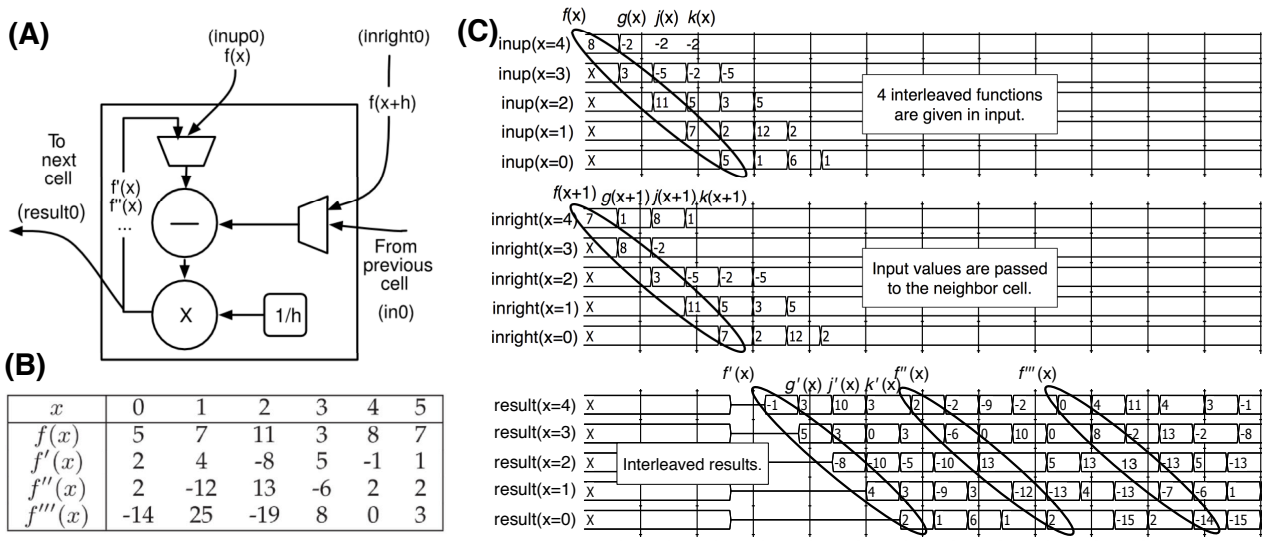


Fig. 7. Systolic array for finite difference derivatives: example of WIL-PT SA. Each cell evaluates the n derivative order of function $f(x)$ in the correspondent point.

5.3.3 Simulation

A simulation of this systolic array has been carried out to demonstrate its correct behavior. Consider the cell structure shown in Fig. 8.A; assume the subtraction and the multiplication require 3 cycles each, the feedback requires 7 cycles and the propagation path to next cell requires 10 cycles. Under these conditions, the time required by the whole feedback loop, given by the feedback itself and the two operations is $T_{loop} = 13$, and the frequency at which inputs can be fed is $K = 3$. Hence, it will be possible to interleave $N = T_{loop}/K = 4$ operations, ensuring a stall after each set of 4 inputs equal to $R = T_{loop}\%K = 1$ cycle. The systolic array simulated has 5 cells, named from 0 to 4 from right to left; therefore, the array is able to evaluate the derivatives of the function given in 5 points. For sake of simplicity, and without leading the generality of the discussion, it is assumed that the distance between values of the sampled function is $h = 1$. It is also worth noticing that the result of multiplication, since it must be fed back in the cell, is truncated, considering only LSBs. The rightmost cell evaluates derivatives in $x = 4$, while the leftmost evaluates derivative in $x = 0$ (of course the interval is arbitrary, the architecture can work for any value of x). $T_p - T_{fb} = 3$, hence the calculation must start from the rightmost cell.

Results shown in Fig. 8.B highlight one of the 4 functions used to test the systolic array for finite difference derivative; values of derivatives in $x = 5$ are known and given from the outside; in Fig. 8.C a simulation is shown. Fig. 8.A shows names of signals for the rightmost cell; one should notice that *in0* is the external input that provides values of derivative of any order in the boundary point; other cells receive from that input the result of neighbor PE. *inup* and *inright* provide values of the function to derive for the evaluation of first order derivative and are used at first step only: indeed in the simulation they assume 4 values only, corresponding to the 4 functions that are evaluated in an interleaved manner. *result0* is the result obtained by the rightmost cell, the one that evaluates derivatives in $x = 4$; its output will be $f'(4)$ when becoming valid, and after other 3 values (that are the value of derivative of the other 3 functions) it is $f''(4)$. The same can be noticed for the other outputs. The delay between results of cells is exactly $T_p - T_{fb} = 3$ cycles, and each new result in one cell can be carried out after $K = 3$ cycles; the delay between the last first-order derivative and the first second-order derivative is instead of 4 cycles because 1 stall must be inserted. *Ctrl* signal, used to select the input of the multiplexers, has the same 3 cycles delay between blocks, which means that it could be treated as a local signal traveling from cell to cell if one is able



x	0	1	2	3	4	5
$f(x)$	5	7	11	3	8	7
$f'(x)$	2	4	-8	5	-1	1
$f''(x)$	2	-12	13	-6	2	2
$f'''(x)$	-14	25	-19	8	0	3

Fig. 8. **(A)** Cell for finite difference derivative: at first step multiplexers are set to output $f(x)$ and $f(x+h)$ to evaluate $f'(x)$; from next step on, it gets the value coming from feedback and the value from previous cell. **(B)** One function and its derivatives used to test the systolic array for finite difference derivative. **(C)** Simulation of the systolic array for finite difference derivative. 4 operations are interleaved to maximize the throughput: input data and results waveforms are shown, and data belonging to $f(x)$ are highlighted.

to design it having propagation time exactly equal to $T_p - T_{fb}$, or, it can be taken from the external and given with an intrinsic delay. It must be set to '1' when the first derivative has been evaluated and the cell is used for evaluating next order derivatives, using the values stored in the cell itself. Since the result is truncated on 5 least significant bits, the values of $f'''(1)$ and $f'''(2)$ are incorrect, but yet valid if truncation is accepted.

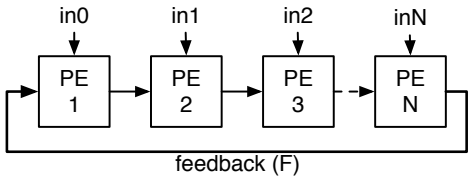


Fig. 9. WEL SA: the output of last PE is fed into the first one. Each PE has a delay of d_i cycles, $i = 1 \dots N$, while feedback has a delay of F cycles.

5.4 WEL Systolic Arrays

The analysis carried out for WIL Systolic Arrays can be used to optimize WEL SAs as well. Referring to Fig. 9, forward loop is made of PEs in the line, each with a delay d_i $i = 1 \dots N$.

$$T_{ff} = \sum_{i=1}^{N} d_i$$

Feedback loop has a delay of $T_{fb} = F$ cycles. Inputs can be provided to each PE with the same rule derived for WIL Systolic Arrays: T_{loop}/K operations can be interleaved where $T_{loop} = T_{ff} + T_{fb} = \sum_{i=1}^{N} d_i + F$ and $K = max\{d_i\}$.

6 Results on CMOS technology

In order to evaluate the improvements achieved by interleaving, the Cell Updates Per Second (CUPS) parameter can be computed. This is a common parameter [3] to evaluate performance of SAs. Often this value is approximated to the maximum achievable CUPS, given by the number of PEs in the array times the clock frequency. In this article we derive instead rigorous equations that can be used to evaluate actual CUPS and predict the effects of pipeline interleaving with more accuracy. In a first step we evaluate the results that can be achieved using as a target a standard CMOS technology.

In a systolic array, PEs rhythmically compute and pass data through the system [50]. Timing of operations follow a "wavefront" order; considering Fig. 3.B, the top-left PE is the one that starts operating first, while the bottom-right one is the last. Given a finite number of inputs, the bottom-right PE is also the one that finishes computation later; hence, total time will be given by the time at which this PE will finish to compute the last result. Total time, T_{end}, is then given by the time for last inputs to reach last PE, plus the time to execute the operation inside the PE itself, called T_{cell}. In the following we will analyze the case of WOIL SA and WIL SA, considering in particular the case of matrix multiplication. For both of them we evaluate the total time of computation and then the CUPS. The concept of "frequency increase" is used throughout the two cases. WEL SA is not detailed since equations are the same of the WIL SA case.

Consider each PE made with two arithmetic blocks, for example a multiplier and an adder, in case of matrix multiplication. They usually have different delay, say T_A

and T_B, $T_A > T_B$, and the frequency f_{clk} is set according to the slowest of the two blocks: $f_{clk} = 1/T_A$. In this way each of the blocks require 1 clock cycle to complete its operation. However, it is possible to set the frequency on the fastest of the two blocks: $f_{clk} = 1/T_B$; the fastest block will need 1 cycle to complete its operation, while the other needs $\lceil T_A/T_B \rceil$ (upper integer). Of course, input data cannot be given at every clock cycle, but after $\lceil T_A/T_B \rceil$ one from the other.

6.1 WOIL Systolic Arrays results

In a WOIL Systolic Array, such as the one in Fig. 3.A, vertical and horizontal propagation must be differentiated. Horizontal propagation of inputs is achieved through shift registers of length L; vertical propagation of partial results depends on the computational time of the cell, hence it is given by the delay of the path P, that is called D according to Fig. 4.A. Call $T_{end}^{(i)}$ the time at which computation of i-th result is available. Then: $T_{end}^{(1)} = (N-1)L + (N-1)D + T_{cell}$, and if we consider last input p:

$$T_{end} = T_{end}^{(p)} = (N-1)L + (N-1)D + (p-1)K + T_{cell} \quad (6)$$

Formula (6) expresses the total time needed to execute operations on a $N \times N$ SA that receives p successive data from each input path. During this period of time each cell will execute p operations (one every time a new input is received). Then, the total cell updates are pN^2, and, given the clock frequency in ns, GCUPS (Giga Cell Updates Per Second) can be evaluated as:

$$GCUPS = \frac{f_{clk} \times pN^2}{(N-1)L + (N-1)D + (p-1)K + T_{cell}} \quad (7)$$

This formula must be adapted in two cases: when the array is used without interleaving operations, and when interleaving is exploited to achieve an improvement in performance. Consider the case of matrix multiplication: $C = A \times B$ where A, B, and C are $N \times N$ matrices. The PE is the one shown in Fig. 4.B. The array is made of $N \times N$ PEs; in case of no-interleaving it is used to evaluate elements of one resulting matrix; in case of n-interleaving instead, it is used to evaluate elements of n different resulting matrices. In case of no-interleaving formula (7) can be adapted considering $p = N$. In case of n-interleaving instead, each cell will update n times more than the previous case, hence $p = nN$; this increase reflects also at the denominator of formula (7) as an increase in total time.

Hereinafter the following notation is used: D_{block} represents the delay of the block expressed in ns; T_{block} represents the delay of the block expressed in clock cycles. K is the number of clock cycles that occur between one input and the successive one, both in the no-interleave version and in the interleaved one.

Results were evaluated for different level of interleave for the example of matrix multiplication, considering inputs at 16-bit and results in 32-bit. In CMOS, we have considered a 45-nm technology, using a 16-bit Wallace multiplier [51] with delay $D_{mul} = 12.65\,ns$ and a 32-bit ripple carry adder with delay $D_{add} = 6.14\,ns$. In case of no-interleave (no-i), without exploiting frequency increase, the clock period must be set to $t_{clk} = 12.65\,ns$ and the number of clock cycles required by each of the blocks is 1. One register is inserted in the propagating path, hence $D = 2$ (number of cycles to pass through the adder and the registers). When inputs are not interleaved, data are passed to the structure when it has finished previous computation, hence $K = T_{mul} + T_{add} + T_{reg} = 3$. $T_{cell} = T_{mul} + T_{add} = 2$. L does not impact the discussion, and can be set to 1. Formula (7) with all parameters set as described reduces to $GCUPS = f_{clk}N^3/(6N-4)$. In case of $(n-i)$ interleave, we can set the frequency according to the fastest block: $t_{clk} = 6.4\,ns$ can be considered in such a way that the multiplier requires 2 cycles to complete its operation, while the adder requires only one: so, $K = 2$. The delay of the register chain to pass result to following cell depends on the number of interleaved operations we want to achieve. If 5 operations must be interleaved for example, D must be equal to 10 ($D = n \times K$), and since adder requires 1 cycle, 9 registers must be inserted: $T_{reg} = nK - T_{add} = 2n-1$. $T_{cell} = T_{mul} + T_{add} = 3$ and again $L = 1$. Formula (7) with all parameters set as described reduces to $GCUPS = f_{clk}nN^3/[2n(2N-1)+N]$.

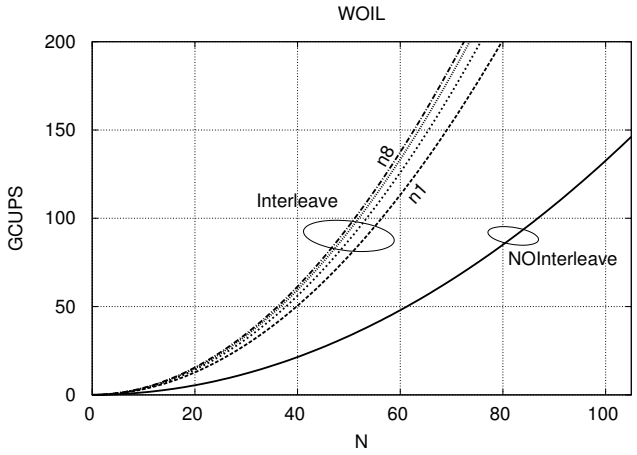


Fig. 10. The effect of interleaving in terms of GCUPS: for the same $N \times N$ WOIL-S SA, interleaving and increasing frequency allow achieving better results.

Fig. 10 shows the GCUPS depending on number of cells N. It is clear that better results can be achieved increasing frequency and using deep interleaves; just increasing frequency and exploit pipelining, still with $n = 1$ we have an improvement with respect to the original case. The improvement saturates with the increase of n; this is due to the fact that an increase in n has an impact in every cell as an increased delay between blocks, that could eventually match the benefits of the gain in the number of evaluated results.

6.2 WIL Systolic Arrays results

Consider a WIL-S Systolic Array; in this case inputs for last PE are inputs to the whole array that have been shifted through registers. L was previously defined as the number of cycles needed for an input to pass through a cell, that is the number of registers of the shift chain. In this case it is possible to assume that this value is equal for left to right or top to bottom transmission. First inputs will be available at last cell after $2(N-1)L$, where $2(N-1)$ is the Manhattan distance between first cell and last one in an array of $N \times N$ PEs. Following formula results: $T_{end}^{(1)} = 2(N-1)L + T_{cell}$. K is the delay between one input and the following one; if we have p inputs, then we can write:

$$T_{end} = T_{end}^{(p)} = 2(N-1)L + (p-1)K + T_{cell} \tag{8}$$

Equations (8) and (6) correspond in case $D = L$.

Following formula expresses GCUPS in the case of WIL-S SA:

$$GCUPS = f_{clk} \frac{pN^2}{2(N-1)L + (p-1)K + T_{cell}} \tag{9}$$

This equation must be adapted in two cases: when the array is used without interleaving operations, and when interleaving is exploited to achieve an improvement in performance. In case of no-interleaving equation (9) can be adapted considering $p = N$. In case of n-interleaving instead, each cell will update n times more than the previous case, hence $p = nN$; this increase reflects also at the denominator of equation (9) as an increase in total time.

Results were evaluated for different levels of interleave for the example of matrix multiplication, considering 16-bit inputs and 32-bit results. As stated before, in CMOS, 45-nm technology was used, having $D_{mul} = 12.65\,ns$ (Wallace multiplier [51]); $D_{add} = 6.14\,ns$. In case of no-interleave (no-i), without exploiting frequency increase, the clock period must be set to $t_{clk} = 12.65\,ns$ and the delay for each of the block is 1, meaning $T_e = T_{ff} = 1$. One register is inserted in the feedback path, $T_{fb} = 1$. $K = T_{loop} = T_{ff} + T_{fb} = 2$ and $T_{cell} = T_e + T_{ff} = 2$ cycles. L does not impact on the discussion, and can be set to 1.

Formula (9) with all parameters set as described reduces to $GCUPS = f_{clk}N^3/(4N-2)$. If instead $(n-i)$ interleaved is used, the frequency can be set according to the fastest block:$t_{clk} = 6.4\,ns$ can be considered in such a way that the multiplier requires 2 cycles to complete its operation ($T_e = 2$), while the adder requires only one($T_{ff} = 1$): so, $K = 2$. The delay of the feedback path depends on the number of interleaved operations that is necessary to achieve: since $n = T_{loop}/K$, being $T_{loop} = T_{ff} + T_{fb}$, it is clear that relation (10) holds:

$$T_{fb} = nK - T_{ff} \tag{10}$$

that in this case is: $T_{fb} = 2n-1$. Also, $T_{cell} = T_e + T_{ff} = 3$, and again $L = 1$. Formula (9) with all parameters set as described reduces to $GCUPS = f_{clk}nN^3/[2N(n+1)-1]$. The advantage of interleaving is shown in Fig. 11. The higher the number of PEs in the array, the higher is also the increase, in terms of GCUPS, that is achieved using interleaving. However, the increase in performance saturates, and after a certain point, higher values of p do not translate in further significant increase in performance.

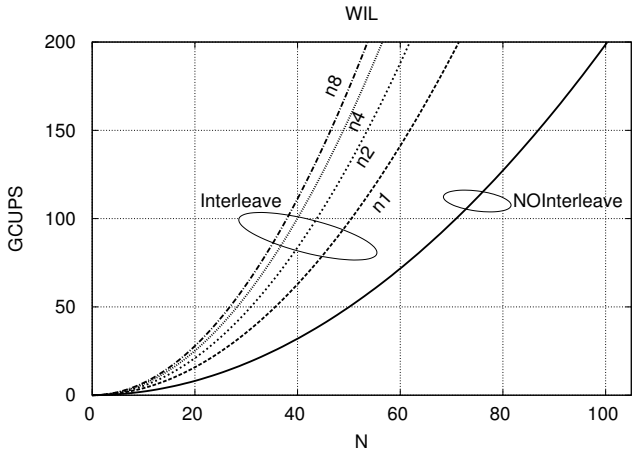


Fig. 11. The effect of interleaving in terms of GCUPS: for the same $N \times N$ WIL-S SA, interleaving and increasing frequency allow achieving better results.

One particular case is interleave with $n = 1$; in this case there is not an interleave inside the loop (i.e. there are not 2 values traveling along the loop together) but still the pipelined nature of the PE and the increase frequency concept are exploited to improve performances. Frequency not always can be increased; however, interleaving can still be exploited to improve performance. In this case t_{clk} is set according to the slowest of the two blocks; all the blocks will then need 1 cycle to complete, hence $K = 1$ and $T_{cell} = 2$. Fig. 12 shows that there is still an increase in performance, however much smaller than in the case of interleaving with frequency increase (notice for example the difference between interleave 8 and interleave fast (that is done with $n = 8$) where the second is with frequency increased).

6.3 Case Study: Protein Alignment Systolic Array

The purpose of this article is to provide a rigorous description of the technique to apply pipeline interleaving to systolic arrays and to introduce metrics to evaluate the effects that can be obtained. In order to suggest a context to the reader, but without the aim of being exhaustive, we provide here a short introduction to a case study we developed to show the benefits of this technique [30]. Protein comparison is gaining importance year after year because it helps biologists in finding correlation between different species, or genetic mutations that can lead to cancer and genetic diseases. Protein sequence alignment is the most computational intensive task when performing protein comparison. For this reason SAs are

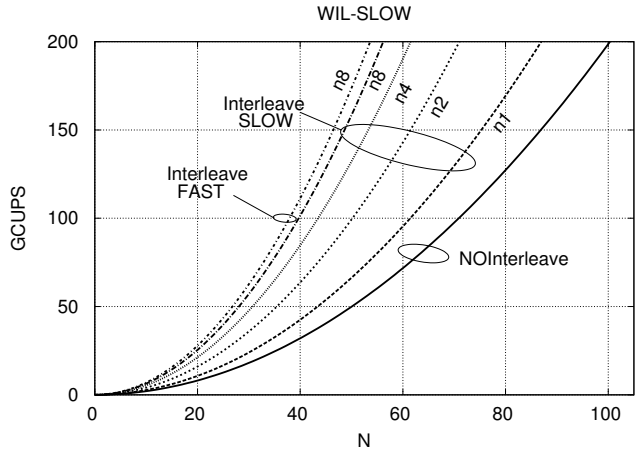


Fig. 12. The effect of interleaving in terms of GCUPS, without increasing frequency: for the same $N \times N$ WIL-S SA, interleaving allow achieving better results; "Interleave fast" is the same array with $n = 8$ and exploiting increase of frequency.

often used to speed-up the execution of these algorithm. In [5] a SA for the Smith-Waterman algorithm has been proposed. According to our classification, this is a WIL-S SA. The optimization with interleaving in CMOS relies on the reduction of critical paths introducing additional registers, and on data interleaving to avoid throughput reduction. We focus on further details about this specific optimization in another paper [30], based on the present one for what concerns the analytic discussion. Focusing on details here would change the nature of this paper. Nevertheless, we can summarize that in CMOS it is possible to achieve a speed-up of 2.5X without affecting area or power requirements significantly. Moreover a NML implementation has been proposed in [52]. The article shows how it is possible to interleave data also in the case of NML to increase throughput without changing the hardware structure. As described before, this allows to avoid retiming (that must take into account also the physical layout of the circuit) and yet obtain improvements using interleaving only. In [52] results show that CMOS circuit can operate at $370\,MHz$ consuming $0.72\,mW$, while for NML the operating frequency is given by the technology and is $100\,MHz$ and power consumption in the case of Magnetoelastic implementation is of $10\,\mu W$ only. It is important to highlight that in NML without applying interleaving there must be a delay of 209 clock cycles between one input and the successive one, while with interleaving it is possible to provide one input at every clock cycle as it is in CMOS: this therefore proves the benefits that derive from the adoption of pipeline interleaving in this technology.

7 Results on Emerging Nanotechnologies

We will now give an overview of the advantages that can be achieved by exploiting interleaving and optimization mechanisms to SAs implemented with new technologies.

TABLE 2
NML WIL-S SA for matrix multiplication

	K	T_{cell}	$CUPS$
no-interleave	38	32+19=51	13447.8 f_{clk}
interleave	19	38 + 19 = 57	26887 f_{clk}

As stated before, this analysis particularly fits the case of Quantum-dot Cellular Automata or N-FET PLA-like circuits, where the delay of wires can be quite long. Designers have put a big effort in trying to reduce it [11]; our analysis allow designers to relax on the layout of the cell and the delay of the wires, exploiting their intrinsic pipelined nature; an analysis of the layout then allows to choose the best interleaving to optimize the usage of each PE. We focus on the WIL-S Systolic Array for matrix multiplication; the structure of the PE is shown in Fig. 6. Assume 16-bit inputs and 32-bit output. Considering for example NML, a suitable adder can be a Carry Lookahead Adder (CLA) whose delay is $T_{ff} = 19$ clock cycles [53], while multiplier [53] has a delay of $T_e = 32$ clock cycles, but can be pipelined. The delay of the feedback loop can be considered equal to the delay of the adder, so $T_{fb} = 19$ can be assumed. In order to match delays, the multiplier can be delayed to reach 38 cycles. Inputs can be given every $K = 19$ cycles, and 2 operations can be interleaved. The delay to transmit data through a PE is at least equal to the forward path (usually bigger), and $L = 20$ can be assumed. The analysis is carried out for a 1024×1024 SA and results are shown in Table 2: a speed-up of nearly 2 is achieved. Notice that in this case the frequency is the same for both solutions, because frequency in QCA depends on technological limitations and cannot be set freely.

In NanoWire Field Effect Transistor (NWFET) technology, the delay of the ripple carry adder is given by the delay of the full adder, and in case of a 32-bit addition it is $33 \times t_{FA}$, while the delay of the Wallace multiplier is $68 \times t_{FA}$ but it can be pipelined. The delay of the feedback loop can be assumed to be equal to that of the adder; hence $K = 34$ (in order to have $T_{mul} = 2$), and again $L = 20$ can be assumed. A comparison of the interleaving method here proposed in different technologies and at different frequencies is reported in Table 3.

Results in Table 3 show that, considering the same frequency ($156\,MHz$, a frequency obtained from the FPGA implementation of the SA), the throughput with CMOS is much better. This is caused by the presence of feedback signals inside the circuit as explained in Section 2. Using interleaving the throughput is greatly increased but still lower than CMOS circuits. However considering the theoretical frequency that these new technologies can reach, the results are outstanding compared to CMOS circuits as shown in Table 3. For Molecular QCA (M-QCA) this estimation was done adopting the best knowledge on this technology available at the moment. Since M-QCA is still under development, these numbers might

change when a realistic layout will be implemented. Nevertheless, the example is useful to highlight potential benefits that can be achieved using pipeline interleaving also with these new perspective technologies, in case they will reach in the future a sufficient level of technological maturity. This means that, in each line of Table 3, moving towards higher levels of interleave, GCUPS always increase.

It must be also clear that even if the systolic processor can run at that frequency, memories and I/O structures should be capable of reaching these speeds as well. A full analysis of the design and speed of memory and I/O structures is beyond the scope of this article, so these results can be considered a "best case analysis".

TABLE 3
Comparison of performance of WIL-S SA for matrix multiplication with different technologies.

GCUPS					
Tech.	f_{clk}	Interleave level			
		No	1	2	3
CMOS		20732	40970	54622	61447
NML	$156\,MHz$	2101	2777	4201	5067
NWFET		1545	2213	3033	3460
	$10\,GHz$	134478	177730	268872	324308
M-QCA	$100\,GHz$	1347782	1777306	2688723	3243083
	$1\,THz$	13477827	17773062	26887237	32430838
NWFET	$312\,GHz$	3086516	4419390	6056905	6910409

Nevertheless, there is one key aspect that can be highlighted. Considering for example NanoWire Field Effect Transistor (NWFET) technology, the evaluation of K gives $K = 34$: this means that a new input must be available every 34 clock cycles, and then the memory can be K times slower than the processor itself, being more easily implementable, or cheaper. If the whole SA reads from one memory only, then a memory with $N \times N$ parallel outputs will be needed; moreover, it will be quite common to have a mismatch between K and the delay of input between successive PEs, that means that reads in memory would be more than one every K cycles; they could reach the maximum of one read every clock cycle, eventually being one read correspondent to $N \times N$ values to fed in the SA.

8 Conclusions

In this article we presented and discussed a systematic method to increase performance of Systolic Arrays. According to the presence of loops in Processing Elements or in lines of the SA, and to the way the results are evaluated (locally inside one PE or through a line) we defined a rigorous taxonomy of types of SAs. Consequently we envisaged specific methods to optimize through *interleaving* the SA usage for each class using practical examples to clarify the explanation. We evaluated in terms of Giga Cell Updates Per Second (GCUPS) the performance improvement both in the case of standard CMOS technology and of emerging technologies (like, for example, QCA and Nanoarrays based on on silicon nanowire FET).

Results encourage the adoption of this method: performance are more than doubled using interleaving $n = 2$, and even better results can be achieved with deeper levels of interleave. In the case of nanotechnologies, whose wires have long delays and are intrinsically pipelined, it is extremely important to adopt this method to compensate the throughput reduction due to feedback signals. Using this resource as optimization technique, designers can trade off between layout constraints and interleaving level.

Our future efforts will concentrate on finding other techniques at algorithm level based on data arrangement to improve the circuit performance and to deal with the massive pipelining that is a feature of many emerging technologies.

References

[1] S. Y. Kung, *VLSI array processors*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[2] J. Brodkin, "10,000-core linux supercomputer built in Amazon cloud," *Network World*, 2011.

[3] T. Rognes, "Faster smith-waterman database searches with inter-sequence SIMD parallelisation," *BMC Bioinformatics*, vol. 12, no. 221, 2011.

[4] M. Graziano, S. Frache, and M. Zamboni, "A hardware viewpoint on biosequence analysis: What's next?" *ACM J. on Emerging Tech. in Computing Systems*, vol. 9, no. 4, 2013.

[5] G. Urgese, M. Graziano, M. Vacca, M. Awais, S. Frache, and M. Zamboni, "Protein alignment HW/SW optimizations," in *2012 19th IEEE Int. Conf. on Electronics, Circuits and Syst. (ICECS)*, 2012, pp. 145–148.

[6] N. Haron and S. Hamdioui, "Why is CMOS scaling coming to an end?" in *Design and Test Workshop, 2008. IDT 2008. 3rd Int.*, dec. 2008, pp. 98–103.

[7] A. Pulimeno, M. Graziano, and G. Piccinini, "UDSM trends comparison: From technology roadmap to ultrasparc niagara2," *IEEE Tran.s on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 7, pp. 1341–1346, july 2012.

[8] L. Lu, W. Liu, M. O'Neill, and E. Swartzlander, "QCA systolic matrix multiplier," in *2010 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, july 2010, pp. 149–154.

[9] M. Niemier, G. Csaba, A. Dingler, X. Hu, W. Porod, X. Ju, M. Becherer, D. Schmitt-Landsiedel, and P. Lugli, "Boolean and non-boolean nearest neighbor architectures for out-of-plane nanomagnet logic," in *2012 13th Int. Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, 2012, pp. 1–6.

[10] X. Ju, M. Niemier, M. Becherer, W. Porod, P. Lugli, and G. Csaba, "Systolic pattern matching hardware with out-of-plane nanomagnet logic devices," *IEEE Tran.s on Nanotechnology*, vol. 12, no. 3, pp. 399–407, 2013.

[11] L. Lu, W. Liu, M. O'Neill, and E. Swartzlander, Jr, "QCA Systolic Array Design," *IEEE Tran.s on Computers*, vol. 62, no. 3, pp. 548–560, 2013.

[12] S. Frache, D. Chiabrando, M. Graziano, F. Riente, G. T. G., and M. Zamboni, "ToPoliNano: Nanoarchitectures design made real," in *2012 IEEE/ACM Int. Symp. on Nanoscale Architectures*. Amsterdam, The Netherlands: IEEE, 2012, pp. 160–167.

[13] M. Graziano, M. Vacca, A. Chiolerio, and M.Zamboni, "An nclhdl snake-clock-based magnetic qca architecture," *IEEE Tran.s on Nanotechnology*, vol. 10, no. 5, pp. 1141–1149, sept. 2011.

[14] M. R. Casu, M. R. Roch, S. V. Tota, and M. Zamboni, "A noc-based hybrid message-passing/shared-memory approach to CMP design," *Microprocessors and Microsystems*, vol. 35, no. 2, 2011.

[15] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum cellular automata," *Nanotechnology*, vol. 4, no. 1, pp. 49–57, 1993.

[16] M. Graziano, M. Vacca, D. Blua, and M. Zamboni, "Asynchrony in Quantum-Dot Cellular Automata nanocomputation: Elixir or poison?" *IEEE Design Test of Computers*, vol. 28, no. 5, pp. 72–83, sept.-oct. 2011.

[17] K. Parhi and D. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters. i. pipelining using scattered look-ahead and decomposition," *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1099–1117, jul 1989.

[18] E. Lee and D. Messerschmitt, "Pipeline interleaved programmable dsp's: Synchronous data flow programming," *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 35, no. 9, pp. 1334–1345, sep 1987.

[19] A. Fernandez, J. Llaberia, J. Navarro, and M. Valero-Garcia, "Transformation of systolic algorithms for interleaving partitions," in *1991. Proceedings of the Int. Conf. on Application Spec, Array Proc.*, sep 1991, pp. 56–71.

[20] S. Frache, M. Graziano, and M. Zamboni, "A flexible simulation methodology and tool for nanoarray-based architectures," in *2010 IEEE Int. Conf. on Computer Design (ICCD)*, oct. 2010, pp. 60–67.

[21] H. Kung, C. Leiserson, and C.-M. U. D. of Computer Science, *Systolic Arrays for VLSI*, ser. CMU-CS. Carnegie-Mellon University, Department of Computer Science, 1978.

[22] H. Lim and J. Swartzlander, E.E., "Multidimensional systolic arrays for the implementation of discrete Fourier transforms," *IEEE Tran.s on Signal Processing*, vol. 47, no. 5, pp. 1359–1370, may 1999.

[23] H. Herzberg and R. Haimi-Cohen, "A systolic array realization of an LMS adaptive filter and the effects of delayed adaptation," *IEEE Tran.s on Signal Processing*, vol. 40, no. 11, pp. 2799–2803, nov 1992.

[24] L.-W. Chang and M.-C. Wu, "A unified systolic array for discrete cosine and sine transforms," *IEEE Tran.s on Signal Processing*, vol. 39, no. 1, pp. 192–194, jan 1991.

[25] H. Yeo and Y. H. Hu, "A modular high-throughput architecture for logarithmic search block-matching motion estimation," *IEEE Tran. on Cir. and Syst. for Video Technology*, vol. 8, no. 3, pp. 299–315, 1998.

[26] Y.-S. Jehng, L.-G. Chen, and T.-D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Tran.s on Signal Processing*, vol. 41, no. 2, pp. 889–900, feb 1993.

[27] S. B. Pan and R.-H. Park, "Unified systolic arrays for computation of the dct/dst/dht," *IEEE Tran.s on Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 413–419, apr 1997.

[28] S. Panchanathan and M. Goldberg, "A systolic array architecture for image coding using adaptive vector quantization," *IEEE Tran. on Cir. and Sys. for Video Technology*, vol. 1, no. 2, jun 1991.

[29] M. Gok and C. Yilmaz, "Efficient cell designs for systolic Smith-Waterman implementations," in *2006. FPL '06. Int. Conf. on Field Programmable Logic and Applications*, aug. 2006, pp. 1–4.

[30] G. Causapruno, G. Urgese, M. Vacca, M. Graziano, and M. Zamboni, "Protein Alignment Systolic Array Throughput Optimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2014.

[31] B. Buyukkurt and W. Najj, "Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs," in *Int. Conf on Field Prog. Logic and App., 2008.*, sept. 2008, pp. 655–658.

[32] W. Jin, C. Zhang, and H. Li, "Mapping multiple algorithms into a reconfigurable systolic array," in *2008. Canadian Conf. on Electrical and Computer Eng.*, may 2008.

[33] D. Rairigh, "Limits of CMOS technology scaling and technologies beyond-CMOS," 2005.

[34] A. Csurgay, W. Porod, and C. Lent, "Signal processing with near-neighborcoupled time-varying quantum-dot arrays," *IEEE Tran. On Circuits and Systems*, vol. 47, no. 8, pp. 1212–1223, 2000.

[35] M. Liu, C. Lent, and Y. Lu, "Molecular electronics - from structure to circuit dynamics," in *Sixth IEEE Conf. on Nanotechnology*. Cincinnati-Ohio, USA: IEEE, 2006, pp. 62–65.

[36] A. Pulimeno, M. Graziano, D. Demarchi, and G. Piccinini, "Towards a molecular QCA wire: Simulation of write-in and read-out systems," *Solid-State Electronics, Elsevier*, vol. 1, p. 7, 2012.

[37] A. Pulimeno, M. Graziano, V.Cauda, A. Sanginario, D. Demarchi, and G. Piccinini, "Bis-ferrocene molecular QCA wire: ab-initio simulations of fabrication driven fault tolerance," *IEEE Tran. on Nanoelectronics*, vol. 12, no. 3, 2013.

[38] M. Vacca, M. Graziano, and M. Zamboni, "Majority voter full characterization for nanomagnet logic circuits," *IEEE Tran. on Nanoelectronics*, vol. 11, no. 5, pp. 940–947, 2012.

[39] M. Graziano, A. Chiolerio, and M. Zamboni, "A technology aware magnetic QCA NCL-HDL architecture," in *International Conference on Nanotechnology*. Genova, Italy: IEEE, 2009, pp. 763–766.

[40] M. Vacca, M. Graziano, and M. Zamboni, "Nanomagnetic logic microprocessor: Hierarchical power model," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, no. 8, p. 8, 2012.

[41] C. Augustine, X. Fong, B. Behin-Aein, and K. Roy, "Ultra-low power nano-magnet based computing: A system-level perspective," *IEEE Tran. on Nanotech.*, vol. 10, no. 4, pp. 778–788, 2011.

[42] M. Awais, M. Vacca, M. Graziano, M. R. Roch, and G. Masera, "Quantum dot cellular automata check node implementation for LDPC decoders," *IEEE Tran.s on Nanotechnology*, vol. 12, no. 3, 2013.

[43] M. Vacca, M. Graziano, and M. Zamboni, "Asynchronous solutions for nano-magnetic logic circuits," *ACM J. on Emerging Technologies in Computing Systems*, vol. 7, no. 4, December 2011.

[44] S. Frache, L. Amaru, M. Graziano, and M. Zamboni, "Nanofabric power analysis: Biosequence alignment case study," in *2011 IEEE/ACM Int. Sym. on Nanoscale Architectures*, 2011, pp. 91–98.

[45] M. Crocker, M. Niemier, and X. S. Hu, "A reconfigurable PLA architecture for nanomagnet logic," *J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 1, pp. 1:1–1:25, Feb. 2012.

[46] M. Crocker, X. Hu, and M. Niemier, "Design and comparison of NML systolic architectures," in *2010 IEEE/ACM Int. Symposium on Nanoscale Architectures*, june 2010, pp. 29–34.

[47] W. Liu, L. Lu, M. O'Neill, E. Swartzlander, and R. Woods, "Design of quantum-dot cellular automata circuits using cut-set retiming," *IEEE Tran.s on Nanotechnology*, vol. 10, no. 5, pp. 1150–1160, 2011.

[48] K. Parhi and D. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters. ii. pipelined incremental block filtering," *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1118–1134, jul 1989.

[49] E. Lee and D. Messerschmitt, "Pipeline interleaved programmable dsp's: Architecture," *IEEE Tran.s on Acoustics, Speech and Signal Processing*, vol. 35, no. 9, pp. 1320–1333, sep 1987.

[50] S.-Y. Kung, K. Arun, R. Gal-Ezer, and D. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications," *IEEE Tran.s on Computers*, vol. C-31, no. 11, pp. 1054–1066, nov. 1982.

[51] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Tran.s on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, feb. 1964.

[52] J. Wang, M. Vacca, M. Graziano, M. RuoRoch, and M. Zamboni, "Biosequences analysis on nanomagnet logic," in *2013 Int. Conf. on IC Design Technology (ICICDT)*, 2013, pp. 131–134.

[53] H. Cho and E. Swartzlander, "Adder and multiplier design in quantum-dot cellular automata," *IEEE Tran.s on Computers*, vol. 58, no. 6, pp. 721–727, june 2009.

Giovanni Causapruno Giovanni Causapruno received the Dr.Eng. degree in Electronics Engineering from Politecnico di Torino, Torino, Italy, in 2012, where he is currently a PhD candidate in Electronics and Communications Engineering. His research interests include parallel processing architectures for nanotechnologies.

Marco Vacca Marco Vacca received the Dr. Eng. degree in electronics engineering from the Politecnico di Torino, Turin, Italy, in 2008. In 2013, he got the Ph.D. degree in electronics and communication engineering. He is currently working as an Research Assistant in the Politecnico di Torino. Since 2010, he has been teaching Design of Digital Circuits and Power Electronics. His research interests include quantum-dot cellular automata and others beyond-CMOS technologies.

Mariagrazia Graziano Mariagrazia Graziano received the Dr.Eng. degree and the Ph.D in Electronics Engineering from the Politecnico di Torino, Italy, in 1997 and 2001, respectively. Since 2002 she is a researcher and since 2005 Assistant Professor at the Politecnico di Torino. Since 2008 she is adjunct Faculty at the University of Illinois at Chicago. Her research interests include design of CMOS "beyond CMOS" devices, circuits and architectures. She is author and co-author of more than 90 published works.

Maurizio Zamboni Maurizio Zamboni got his Electronics Eng. degree in 1983 and the Ph. D. degree in 1988 at the Politecnico di Torino. He joined the Electronics Department of the Politecnico di Torino in 1983, became Researcher in 1989, Associate Professor in 1992 and Full Professor of Electronics in 2005. His research activity focuses on multiprocessor architectures design, in IC optimization for Artificial Intelligence, Telecommunication, low-power circuits and innovative beyond CMOS technologies. He is co-author of more than 120 scientific papers (three invited) and three books.